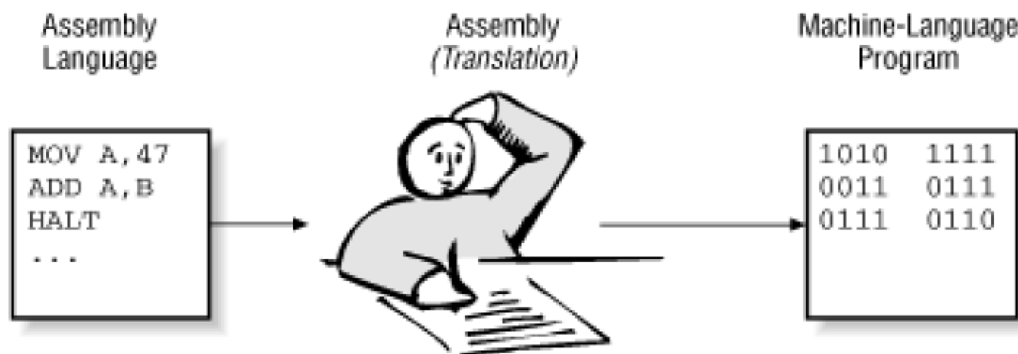
Вовед во програмскиот јазик C
Introduction to programming in C Вовед во програмскиот јазик C

Општ развој на програмските јазици

Со појавата на првите компјутери се појавува потребата од пишување на програми преку кои ќе функционираат деловите од компјутерот. Во тоа време програмерите програмираа на многу неприроден начин, комбинирајќи низи од 0 и 1, користејќи го т.н. **машински јазик**.

Подоцна програмерите согледуваат дека од секој блок на 0 и 1 може да се искомбинираат наредби-зборови кои ќе се многу поразбирливи и полесно ќе се применуваат во текот на пишувањето на програмите. Ваквиот програмски јазик е познат како **assembler.**



Со тек на времето се појавуваат т.н. **програмски јазици на високо-ниво**, кои му нудат на програмерот множество на инструкции кои се лесно разбирливи, а исто така се доволно прецизни и едноставни за компјутерот да може да ги разбере (овде спаѓаат FORTRAN, COBOL, PASCAL…).

Создаден во 1970 година од програмерот Денис Ричи (Dennis Ritchie) и Брајан Керниган (Brian Kernighan) од лабораториите Бел (AT&T Bell Labs) 1972. Негов предок е јазикот B, развиен од Кен Томсон (Ken Thompson) во 1970 година.

ALGOL 60 -> CPL -> BCPL -> B -> C.







C бил почетно дизајниран за пишување на оперативни системи. Јазикот бил екстремно едноставен и флексибилен, така што тој подоцна се користи за пишување на најразлични програми. Поради овие причини јазикот станува најпопуларен програмски јазик во светот.

Идејата за креирањето на програмскиот јазик C е давањето на слобода на програмерот при организацијата и пишувањето на програмата, односно да го напише кодот (програмот) на начин кој е разбирлив за него, а и за останатите програмери. По пишувањето на програмата се користи **компајлер** кој ја преведува програмата во машински код кој е лесно разбирлив за компјутерот.



Основни елементи на програмскиот јазик C

- Азбука на јазикот: големи и мали англиски букви, арапски цифри и интерпункциски знаци

- Секое име содржи само букви, цифри и _.
- Името секогаш започнува со буква.
- Името не смее да содржи празни места и интерпункциски знаци.
- Името не смее да биде еднакво со клучен збор.
- Има разлика помеѓу големи и мали букви.

Секој програмски јазик на високо ниво се состои од множество на резервирани зборови, а комбинацијата од еден или неколку клучни зборови дава наредба од програмскиот јазик.

Множеството на клучни зборови од програмскиот јазик C е следното (32 според ASCII стандардот, 28 според Richie).

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Програмскиот јазик C е функционален јазик, односно кодот напишан во овој програмски јазик е базиран на функции. Основниот облик на програма напишана во програмскиот јазик C е следниот:

```
void main()
{
    deklaracija_na_promenlivi;
    programski_naredbi;
}
```

main е функција која означува главна програма. Во овој специјален тип на функција може да се декларираат променливи, нови под-функции, и да се пишува кодот на програмата. Секоја програма содржи една или повеќе функции, но точно една што се нарекува main. Оваа функција се повикува прва при извршување на С програма

Функциите може да примаат влезни параметри. Ова се обезбедува преку () делот по името на функцијата. Во овој случај празните загради означуваат дека оваа функција не прима влезни параметри.

Типот на резултатот кој го враќа функцијата стои пред името на функцијата. Во овој случај резервираниот збор **void** означува дека функцијата не треба да врати резултат.

Телото на секоја функција започнува со **{**, а завршува со **}**. Наредбите кои се употребуваат меѓусебно се одвојуваат со **;** - точка и запирка.

За дополнително појаснување на некој програмски сегмент многу често се користат коментарите. Во програмскиот јазик С коментарите се задаваат преку користење на **/* … ***/ конструктот или преку **//…** конструктот.

Exercise:

Problem: Програма Dobredojdovte na TMF!

Solution:

#include <stdio.h>

void main()

{

```
printf(“Dobredojdovte na TMF!\n”);

/* na ekran ke se ispecati gornata poraka */

}
```

Опис на кратката програма:

#include

- #include <stdio.h> е претпроцесорска директива – му кажува на компјутерот да ја вметне содржината на определена датотека и понатаму содржините во таа датотека се употребливи во програмата
- <stdio.h> во датотеката се сместени програмските кодови на влезно-излезните операции
- директивата треба да се појави сама и да биде напишана во една линија
- директивите не завршуваат со ;

printf(“Zdravo na site!\n");

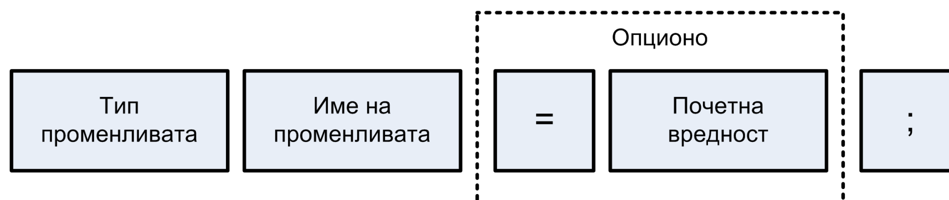
- дава инструкции компјутерот да изврши акција, да ја прикаже на комјутерскиот екран реченицата ограничена со наводниците
- целата линија се нарекува израз
- сите изрази во C мора да завршат со ;
- \ - специјален знак (escape character)
- означува дека printf( )треба да направи нешто различно од вообичаеното однесување
- \n е ознака за знакот нов ред

return 0;

- по конвенција return 0, означува дека програмата завршува успешно со својата работа

Променливи

Променливите претставуваат места во меморијата каде што ќе се чува некоја вредност. Во C ова место во меморијата се идентификува преку името на променливата. Сите променливи кои ќе се користат во програмата треба претходно да бидат декларирани, а потоа да се користат. Начинот на декларација на променливите е следниот:



Типовите на променливите може да бидат:

Цели броеви	Знаковни	Децимални броеви
int	char	float
short		double
long		
unsigned		

При дефинирањето на имињата на променливите треба да се запазат следниве правила, односно имињата на променливите треба да содржат:

- мали букви од **a** до **z**;
- големи букви од **A** до **Z**;
- цифри од **0** до **9**;

- знак за подвлекување _ кој се третира како буква;
- најчесто должината на имињата на променливите е 32 знаци;
- C ги разликува малите и големите букви!

Exercise:

Problem:

Да се напише програма во C која ќе ја пресмета сумата на два броја.

Solution:

```
void main()

{

int a, b, c;

a = 5;

b = 10;

c = a + b; // c=15

}
```

Како што се гледа од примерот освен делот за декларирање на променливите, останатиот дел од програмата ги содржи наредбите кои ја извршуваат бараната задача. Бидејќи се работи за математичка операција, користени се оператори за работа со декларираните променливи.

Оператори

Операторите се користат за градење на изрази, при што операциите се изведуваат од лево надесно со што се применува правилото на приоритет на операторите во нивното изведување.

Аритметички оператори:

Се применуваат над бројни променливи (цели броеви или децимални броеви).

+	собирање на два броја
-	одземање на два броја
*	множење на два броја
/	делење на два броја
%	остаток при делење на два цели броеви

оператор	израз	еквивалентен израз
+=	x+=2;	x=x+2;
-=	x-=2;	x=x-2;
*=	x*=2;x*=a+b;	x=x*2;x=x*(a+b);
/=	x/=2; x/=j+2;	x=x/2; x=x/(j+2);
%=	x%=2;	x=x%2;

Релациони оператори:

Се применуваат над било кои споредливи типови на податоци, а резултатот е цел број 0 или 1. 0 – неточно, 1 – точно.

<	Помало
<=	помало еднакво
>	Поголемо
>=	поголемо еднакво
==	Еднакво
!=	Различно

Логички оператори:

Се користат најчесто во комбинација со релационите оператори за формирање на сложени логички изрази, кои повторно враќаат резултат 0 или 1.

&&	И
||	ИЛИ

!	НЕ

Редослед на извршување на оператори (операции):

- Изразот во загради се пресметува на почеток (има највисок приоритет). Ако заградите се вгнездени, изразот во највнатрешните загради се пресметува прв. Ако постојат повеќе загради на исто ниво во изразот, тие се пресметуваат одлево надесно.
- Операторите *. /, или % (што се однесуваат на операциите множење, делење, модул) се пресметуваат како втори по ред (имаат среден приоритет). Ако во изразот има повеќе операции од овој вид, тие се пресметуваат одлево надесно.
- Операторите + или – (собирање или одземање) се пресметуваат последни. Ако во изразот има повеќе операции од овој вид, тие се пресметуваат одлево надесно.

Самата операција на доделување исто така враќа вредност и може да се вметне во друг израз. Операторот за доделување е пофлексибилен отколку што тоа на прв поглед се чини, така да истиот може да се употреби и на следниот начин:

```
int i, j, k, l, m, n;
i = j = k = l = m = n = 22;
```

n=22 е првата операција што се извршува, и тоа прави вредноста 22 да биде расположива за следната операција додели ја вредноста 22 на променливата m, итн.

Приоритет

- Во принцип сите унарни оператори имаат повисок приоритет од бинарните
- Употребата на загради го менува приоритетот
- Во C се дефинирани 15 нивои

Асоцијативност

1. За два оператора со ист приоритет, операцијата што треба да се изврши се избира на основа на правилата за асоцијативност на операторите
2. Дефинирани се “одлево надесно” и “оддесно налево”

Печатење на податоци (излезни функции)

Како што беше покажано на првиот пример “Dobredojdovte na TMF!”, се користи функција за испраќање на некој текст кон стандардниот уред за испис на податоци – мониторот.

Бидејќи C не вклучува наредби за влез и излез на податоци, се користи библиотека со функции која претходно мора да се пријави во програмата. Тоа се прави со користење на:

#include<stdio.h>

stdio – standard input output

На овој начин пред да се искомпајлира програмата, предпроцесорот на C знае дека треба да ги вклучи функциите од библиотеката stdio.h со цел да не се јави грешка при компајлирањето. За печатење се користи функцијата printf:

printf(kontrolna_niza,lista_na_promenlivi);

Контролната низа содржи било каков текст за испис и контролни знаци предводени од % или \. Контролните знаци зависат од видот на променливата чија вредноста треба да се испише или од саканата акција што треба да биде превземена.

Во следната табела се прикажани сите контролни низи:

контролна низа	објаснување
%d	За цели броеви
%i	За цели броеви
%c	За знаци
%s	За низа од знаци
%e	реален број во технички формат (e)
%E	реален број во технички формат (E)
%f	реален број во децимален формат
%g	реален број во пократкиот од форматите %e и %f
%G	реален број во пократкиот од форматите %E и %f
%u	цел број без предзнак
%o	октален цел број без предзнак
%x	хексадецимален цел број без предзнак (мали букви)
%X	хексадецимален цел број без предзнак (големи букви)
%p	Прикажува покажувач
%n	бројот на испишани знаци се доделува на аргументот

%%	испишување на знакот %

Во низата на променливи покрај променливи може да има и константи и аритметички изрази.

Example:
Примена на printf функцијата.

```
#include<stdio.h>
void main()
{
    printf(“, brojot na znaci e
%d”,printf(“abcd”));
}
```

Оваа програма на екранот ќе испечати

abcd, brojot na znaci e 4

Задачи

Exercise:

Problem:

Да се напише програма која ќе ја пресметува вредноста на математичкиот израз: x = 3/2 + (5 – 46*5/12)

Solution:

```
#include <stdio.h>

void main()
```

```
{
float x;
x = 3/2 + (5-46*5/12);
printf(“Vrednosta na x e %f\n”, x);
}
```

Exercise:

Problem:

Да се напише програма која за вредноста 25 на x ќе го пресмета и испечати неговиот квадрат.

Solution:

```
#include <stdio.h>
void main()
{
int x, kvadrat;
x = 25;
kvadrat = x*x;
printf(“%d na kvadrat e %d.\n”, x, kvadrat);
}
```

Exercise:

Problem:

Да се напише програма која за дадени страни на еден триаголник и ќе ги испечати периметарот и квадратот од плоштината (нека се работи со a=7, b=7.5, c=13.2).

Solution:

```
#include <stdio.h>

void main()

{

float a=7.0, b=7.5, c=13.2;

float L, P, s;

L = a + b + c;

s = L/2;

P = s*(s-a)*(s-b)*(s-c);

printf("Plostinata na kvadrat e: %f\n", P);

printf("Perimetarot e: %f\n", L);

}
```

Exercise:

Problem:

Да се напише програма која за дадени страни на еден правоаголник ќе ги испечати неговите плоштина и периметар (пр. a=7, b=10).

Solution:

```
#include <stdio.h>

void main()

{

int a, b;

int L, P;

printf("Vnesete gi stranite na pravoagolnikot:")

scanf("%d %d", &a, &b);

L = 2*a + 2*b;

P = a * b;

printf("Plostinata e: %f\n", P);

printf("Perimetarot e: %f\n", L);

}
```

Exercise:

Problem:

Да се напише програма за пресметување на аритметичката средина на броевите 3, 5 и 12.

Solution:

```
#include <stdio.h>

void main()

{

int a=3;
```

```
int b=5;

int c=12;

float as;

as = (a + b + c)/3;

printf("Aritmetickata sredina e: %f\n", as);

}
```

Exercise:

Problem:

Да се напише програма која ќе ги испечати на екран остатоците при делењето на бројот 19 со 2, 3, 5 и 8.

Solution:

```
#include <stdio.h>

void main()

{

int a=19;

printf("Ostatokot pri delenjeto so 2 e: %d\n", a%2);

printf("Ostatokot pri delenjeto so 3 e: %d\n", a%3);

printf("Ostatokot pri delenjeto so 5 e: %d\n", a%5);

printf("Ostatokot pri delenjeto so 8 e: %d\n", a%8);

}
```

Exercise:

Problem: Пример со податочни видови

Solution:

```
#include <stdio.h>

int main()

{

int suma;

float pari;

char bukva;

double pi;

suma = 10; /* pridruzi celobrojna integer vrednost */

pari = 2.21; /* pridruzi realna float vrednost */

bukva= 'A'; /* pridruzi znak */

pi = 2.01E6; /* pridruzi realna double vrednost*/

printf(“Vrednost na suma = %d\n", suma );

printf(“Vrednost na pari = %f\n", pari );

printf(“Vrednost na bukva = %c\n", bukva );

printf(“Vrednost na pi = %e\n", pi );

}
```

Излез:

Vrednost na suma = 10

Vrednost na pari = 2.210000

Vrednost na bukva = A

Vrednost na pi = 2.010000e+06

Константи и оператори

Константи

Терминот константа значи дека истата не се менува за време на работата на програмата. Секоја променлива се декларира дека припаѓа на одреден тип податоци. Овој тип ја дефинира големината на променливата и како истата може да се користи. Слично, кога се специфицира константа, се дава и типот на константата. Кај променливите типот е очигледен со нивната декларација. Константите меѓутоа не се декларираат. Одредувањето на нивниот тип не е толку едноставно.

Постојат повеќе типови константи во C:

- Броевите кои содржат „**.**" или „**e**" се **double**: 3.5, 1e-7, -1.29e15
- За наместо double да се користат **float** константи на крајот се додава „F": 3.5F, 1e-7F
- За **long double** константи се додава „**L**": 1.29e15L, 1e-7L
- Броевите без „**.**", „**e**" или „**F**" се **int**: 1000, -35
- За **long int** константи се додава „**L**": 9000000L.

Именуваните константи може да се креираат со користење на резервираниот збор **const**. Со користење на const типот на константата експлицитно се дефинира. По креирањето на константата, таа може да се јави само од десната страна на операторот за доделување вредност.

Example:

```
#include <stdio.h>
main()
{
    const long double pi = 3.141592653590L;
    const int denovi_vo_nedela = 7;
    const nedela = 0; // po default int
```

```
    denovi_vo_nedela = 5; // greska
}
```

Именуваните константи може да се креираат и со користење на предпроцесорот и за нив по правило се користат големи букви.

Example:

```
#include <stdio.h>

#define PI 3.141592653590L
#define DENOVI_VO_NEDELA 7
#define NEDELA 7
main()
{
    long broj = PI;
    int den = NEDELA;
}
```

Бидејќи предпроцесорот е едитор тој ја изведува операцијата најди и замени. За да се влезе во овој начин на работа се користи наредбата **#define**. Нејзината синтакса е всушност:

#define tekst_za_baranje tekst_za_zamena

Се заменуваат само цели зборови. Низи од знаци во наводници се игнорираат.

Оператори

Постојат повеќе групи на оператори и тоа:

- Аритметички оператори (+, -, *, /, % (не работи со реални броеви))
- Релациони оператори (>, <, >=, <=, = =, !=)
- Логички оператори (&&, ||)
- Оператори за инкрементирање и декрементирање (++, --)

Аритметичките оператори се користат за претставување на математички изрази. При тоа, + и – може да се користат и на унарен начин:

```
x = + y;
```

```
x = - y;
```

Првата наредба е иста со x=y, додека втората ја множи вредноста на y со -1, а потоа ја доделува на x.

Реални броеви наспроти целобројни вредности

Операторот за делење е специјален. Постои голема разлика меѓу делењето на целобројни и реални вредности. Ако станува збор за целобројно делење, резултатот е само целиот дел од делењето. Така на пример, 19/10 е 1. Ова значи дека делењето се извршува на различни начини во зависност од контекстот во кој се наоѓа, односно во зависност од типот на операндите со кои ќе работи. Така, ако **двата** операнди се цели броеви ќе се изврши целобројно делење.

Ако барем деленикот или делителот е реален број, тогаш се извршува делење на реални броеви и резултатот од ова делење е реален број. Така 19.0/10.0 е 1.9 (резултатот е исти и за 19/10.0 и 19.0/10).

Example:

```
int main(void)
{
    int i = 5, j = 4, k;
    double f = 5.0, g = 4.0, h;

    k = i / j;  // celobrojno delenje na dva int
vrednosti
    h = f / g;  // realno delenje rezultat 1.25
    h = i / j;  // celobrojno delenje iako h e
realen, rezultatot e 1.000

    return 0;
}
```

Програмскиот јазик C користи различни аритметички оператори и различни типови на операнди така што операторот ќе одбере каков вид на операција ќе изведе. При тоа компајлерот се грижи само за типот на операндите. При изведувањето на операцијата не се зема во предвид типот на променливата на која и се доделува резултатот.

Оператори за инкрементирање и декрементирање

Операторот за инкрементирање (++) го зголемува операндот за еден, додека операторот за декрементирање (--) го намалува операндот за еден. Операндот кој се користи со овие оператори мора да биде една променлива и затоа овие оператори се нарекуваат унарни.

Така, ако на целобројната променлива i и е доделена вредноста 5, изразот ++i ја зголемува вредноста на i за еден, со што i станува 6, додека изразот --i ја намалува вредноста на i за еден и i станува 4.

Операторите за инкрементирање и декрементирање може да се користат на два различни начини, зависно од тоа дали операторот е запишан пред или по операндот.

1. Ако операторот е пред операндот (++i), тогаш операндот ќе ја промени својата вредност пред да се искористи во понатамошниот тек на наредбата.
2. Ако пак операторот е после операндот (i++), тогаш вредноста на операндот се менува откако истата е искористена.

Example:

```
i = 1;
printf(“ i = %d\n”, i);//i=1
printf(” i = %d\n”, ++i);//i=2
printf(” i = %d\n”, i);//i=2
printf(” i = %d\n”, i++);//i=2
printf(” i = %d\n”, i);//i=3
```

Релациони и логички оператори

Треба да се води сметка дека нема boolean (булов) тип на податок во C, наместо него се користи целобројниот int тип. При тоа, вредноста 0 е неточно, додека секоја друга вредност е точно.

Резултатот на логичката И операција (оператор &&) ќе биде точно само ако двата операнди се точни, додека резултатот на логичката операција ИЛИ (оператор ||) ќе биде точно ако било кој од операндите или, пак, двата операнди се точни. Со други зборови, резултатот од логичката операција ИЛИ ќе биде неточно, само ако двата операнди се неточни. Резултатот од логичката операција НЕ (оператор !) ќе биде спротивното од операндот, односно точно ако операндот има вредност неточно, и неточно ако операндот има вредност точно.

При тоа треба да се води сметка дека резултатот на секој релационен логички израз во C е 0 за неточно или 1 за точно (и покрај тоа што секоја ненулева вредност ја третира како точно).

```
i = 5; j = 0; k = -1;
s = i && k || j; // s = 1
```

поради истиот приоритет прво се изведува И операцијата, а потоа ИЛИ.

Example:

```
# include <stdio.h>
main()
{
    float f = 5.5;
    int i = 7;
    char c = ‘w’;

    printf("%d\t",(i>=6)&&(c=='w')); // 1
    printf("%d\t",(i>=6)&&(c==119)); // 1
    printf("%d\t",(f<11)&&(i>100));  // 0
    printf("%d\n",(c != 'p')||((i+f)<=10)); // 1
    // !!! da se upotrebuvaat zagradi
    printf("%d %d\n",(!i == 7), (!(i == 7))); // 0
0
    // da vnimava da ne se koristi = namesto ==
    printf("%d %d\n",(!i == 0), (!(i = 0)));  // 0
1
}
```

Треба да се води сметка дека извршувањето на релационите логички изрази е од лево на десно и дека истата не се изведува докрај доколку нема потреба за тоа.

Оператор за доделување (=)

Доделувањето на вредност се врши со помош на = операторот. При тоа вредноста од левата страна на = се доделува на променливата од десната страна на =. Употребата на овој оператор е многу флексибилна. Доделената вредност е секогаш достапна за повторна употреба при доделување и при тоа операцијата се изведува секогаш од десно на лево, а има низок приоритет:

```
int i, j, k, l, m, n;
i = j = k = l = m = n = 22;
printf(”%i\n”, j=93);
```

Влез и излез на податоци

Влез и излез на податоци

Внесување на податоци со scanf() функцијата

Влезните податоци може да се внесат во компјутерот од стандардниот влез со користење на библиотечната функција scanf(). Обликот на оваа функција е следниот:

scanf(Контролна низа од знаци, arg1, arg2, … , argn)

Контролната низа од знаци е всушност низа од знаци (стринг) кој ја содржи потребната информација за форматирање, а arg1, arg2, ..., argn се аргументите кои ги претставуваат индивидуалните податоци. Аргументите претставуваат покажувачи кои ја даваат мемориската адреса каде ќе се смести податокот кој ќе се прочита. Контролната низа од знаци е изградена од индивидуални групи на знаци со една група на знаци за секој податок кој ќе се чита. Секоја група на знаци мора да започне со знакот %. Во нејзината наједноставна форма една група знаци се состои од % по кој следи знак за конверзија кој го дава типот на соодветниот мемориски елемент каде ќе се запише податокот.

Знак законверзија	**Значење**
c	Податочниот елемент е еден знак
d	Податочниот елемент е децимален цел број
f	Податочниот елемент е реална вредност

Знак законверзија	**Значење**
h	Податочниот елемент е краток цел број
I	Податочниот елемент е децимален, хексадецимален или октален цел број
o	Податочниот елемент е октален цел број
s	Податочниот елемент е низа од знаци по која следи едно празно место, нов ред или табулатор
u	Податочниот елемент е децимален цел број без знак
x	Податочниот елемент е хексадецимален цел број

Example:

```
#include <stdio.h>
main()
{
    char del;
    int delbroj;
    float cena;
    ...
    scanf(“%c%d%f”,&del,&delbroj,&cena);
    ...
}
```

Првата група знаци покажува дека првиот аргумент е знак, втората дека вториот аргумент е децимален цел број, додека третата дека третиот аргумент е реален број.

Со помош на scanf() функцијата вредностите на трите променливи del, delbroj и cena можат да се прочитаат од стандардниот влез кога ќе се изврши програмата. Влезот за програмата може да е следниот:

Pc 12345 570.34

или

Pc

12345

570.34

Example:

```
#include <stdio.h>
main()
{
    int a, b, c;
    ...
    scanf(“%3d %3d %3d”,&a,&b,&c);
    ...
}
```

Влез	**Излез**
1 2 3	a=1, b=2, c=3
123456789	a=123 b=456 c=789

Example:

```
#include <stdio.h>
main()
{
    int i;
    float f;
    char c;
    ...
    scanf(“%3d %5f%c”,&i,&f,&c);
    ...
}
```

Влез	**Излез**
10256.875 T	i=102 f=56.87 c=5

Треба да се напомене дека функцијата scanf го враќа бројот на прочитани параметри.

Example:

```
#include <stdio.h>
main()
{
    int i;
    float f = 3.0;
    char c = ‘a’;

    i = scanf(“%f %c”,&f,&c);
    printf(”%d %f %c”,i, f, c);
}
```

Влез	**Излез**
a c	0 3.0 a
5.0 c	2 5.0 c

Печатење на податоци со printf() функцијата

Излезните податоци може да се запишат од компјутерската меморија на стандардниот излез со користење на библиотечната функција printf. Општата форма на printf функцијата е

printf(Kontrolna niza od znaci, arg1, arg2, …, argn)

каде контролната низа се однесува на низа од знаци која содржи информации за форматирањето. arg1, arg2, …, argn се аргументите кои

ги претставуваат индивидуалните излезни податоци.

Example:

```
#include <stdio.h>
#include <math.h>
main()
{
    float i = 2.0, j = 3.0;
    printf (“%f %f %f %f”, i, j, i+j, sqrt(i+j));
}
```

Оваа програма на екранот ќе печати:

2.000000 3.000000 5.000000 2.236068

Бројот на конверзии во даден формат треба точно да одговара на бројот на аргументи во функцијата. При тоа, С ова нема да го потврди. Ако се дадени поголем број аргументи, вишокот аргументи се игнорираат. Ако, пак, нема доволен број аргументи, С ќе генерира чудни броеви за аргументите кои недостигаат.

Example:

```
#include <stdio.h>
/* Variable for computation results */
int answer;
int main()
{
    answer = 2 + 2;
    printf("The answer is %d\n");
```

```
    return (0);
}
```

Example:

```
#include <stdio.h>
float result; /* Rezultat od delenje */
int main()
{
    result = 7.0 / 22.0;
    printf("Rezultatot e %d\n", result);
    return (0);
}
```

Задачи за вежбање:

ЗАДАЧА 1.

Да се напише програма за пресметување и печатење на плоштината и периметарот на круг која ќе чита реален број кој го претставува радиусот на кругот.

ПРОГРАМА:

```
#include <stdio.h>

int main(void)

{

long double radius = 0.0L;
```

```
long double plostina = 0.0L;

const long double pi = 3.1415926353890L;

printf("vnesi radius na krugot ");

scanf("%Lf", &radius);

plostina = pi * radius * radius;

printf("P na krug so r %.3Lf e %.12Lf\n", radius, plostina);

return 0;

}
```

ЗАДАЧА 2.

Да се напише програма која чита големи букви од тастатура и ги печати истите како мали букви.

ПРОГРАМА:

```
#include <stdio.h>

int main(void)

{

char ch;

printf("Vnesi golema bukva ");

scanf("%c", &ch);

printf("Mala ekvivalent bukva na '%c' e '%c'\n", ch, ch-'A'+'a');

return 0;
```

}

ЗАДАЧА 3.

Кој ќе биде излезот на следната програма?

```
#include <stdio.h>

int main(void)

{

int i=0, j, k=7, m=5;

j = m += 2;

printf(“j = %d\n“, j); // j = 7

j = k++ > 7;

printf(”j = %d\n”, j); // j = 0

j = i == 0 || --k;

printf(“j = %d\tk = %d\n”, j, k); // j = 1 k = 8

return 0;

}
```

ЗАДАЧА 4.

Да се напише програма која ќе прочита два цели броја и ќе ја испечати нивната сума, разлика, производ и остатокот при делењето. Програмата исто така ќе прочита и со која прецизност треба да ги испечати двата броја.

ПРОГРАМА:

```
#include <stdio.h>

int main(void)

{

int prv, vtor, vkupno, dec;

printf(“vnesi dva broja “);

scanf(”%i %i”, &prv, &vtor);

printf(“vnesi preciznost vkupno decimalni mesta “);

scanf(”%i %i”, &vkupno, &dec);

printf(“%i + %i = %0*i\n“, prv,vtor,vkupno,prv + vtor);

printf(“%i - %i = %0*i\n“, prv,vtor,vkupno,prv - vtor);

printf(“%i * %i = %0*i\n“, prv,vtor,vkupno,prv * vtor);

printf(“%i / %i = %0*.*lf\n“,prv,vtor,vkupno,dec,prv / vtor);

printf(“%i %% %i = %0*i\n“, prv,vtor,vkupno,prv % vtor);

return 0;

}
```

ИЗЛЕЗ:

```
vnesi dva broja 12345 56789

vnesi preciznost vkupno mesta decimalni mesta 9 3

12345 + 56789 = 000069134
```

12345 - 56789 = -00044444

12345 * 56789 = 701060205

12345 / 56789 = 00000.000

12345 % 56789 = 000012345

Вежба-1
Воведување на концептите за константи и променливи и структура на програма во програмскиот јазик C.

Програмски јазик C - ВЕЖБА 1

Почетни задачи

1. Структура на програма во C

Example:

```
int main()
{
        //telo na programata
}
```

main() – главна програма //telo na programata – место каде се пишуваат командите кои сакаме компјутерот да ги изврши. Истите се одделуваат со знакот ;

2. Вклучување на библиотеки (библиотечни функции)

Example:

```
#include<stdio.h>
int main()
```

```
{
}
```

Со наредбата #include<stdio.h> ги вклучуваме функциите за приказ на екран и внес од тастатруа во нашата програма.

3. Приказ на екран со користење на функцијата printf()

Example:

```
#include<stdio.h>
int main()
{
        printf(“Programski jazik C”);
        system(“pause”);
}
```

Доколку се изврши горенаведената програма на екран се печати:
Programski jazik C

4. Приказ на екран со користење на функцијата printf()

Example:

```
#include<stdio.h>
int main()
```

```
{
        printf(“Po zavrsuvanje na semestarot site
ke znaeme”);
        printf(“ da programirame vo programskiot
jazik C”);
        system(“pause”);
}
```

Доколку се изврши горенаведената програма на екран се печати: Po zavrsuvanje na semestarot site ke znaeme da programirame vo programskiot jazik C

5. Концепција на променлива

Example:

```
#include<stdio.h>
int main()
{
        int a;
        a=16;
        printf(“vrednosta na promenlivata a e %d”,
a);
        system(“pause”);
}
```

Со наредбата int a; се дефинира променлива од целоброен тип (int). Таа се иницијализира со вредноста 16 со помош на наредбата a=16;.

Со наредбата printf(“vrednosta na promenlivata a e %d”, a); на екран се испишува:

vrednosta na promenlivata a e 16

Ако се направи промената:

Example:

```
#include<stdio.h>
int main()
{
     int a;
     a=3;
     printf(“vrednosta na promenlivata a e %d”,
a);
     system(“pause”);
}
```

вредноста на променливата а во овој случај ќе биде 3 и на екран ќе се испечати:

vrednosta na promenlivata a e 3

6. Концепција на константа

Example:

```
#include<stdio.h>
int main()
```

```
{
     const int k = 7;
     printf(“vrednosta na konstantata k e %d”, k);
     system(“pause”);
}
```

Со наредбата const int k; се дефинира константа од целоброен тип (int). Нејзината вредност е 7.

Со наредбата printf(“vrednosta na konstantata k e %d”, k); на екран се испишува:

vrednosta na konstantata k e 7

Вредноста на констатите неможе да се менува. Ако се напише следната програма:

Example:

```
#include<stdio.h>
int main()
{
     const int k;
     k=8;
     printf(“vrednosta na konstantata k e %d”, k);
     system(“pause”);
}
```

Програмата ќе јави грешка (вредноста на констатите неможе да се менува).

7. Збир на две променливи

Верзија 1

Example:

```
#include<stdio.h>
int main()
{
     int a, b;
     a=10;
     b=12;
     printf(“zbirot na promenlivite a i b e %d”,
a+b);
     system(“pause”);
}
```

Оваа програма на екран ќе испечати:

zbirot na promenlivite a i b e 22

Верзија 2

Example:

```
#include<stdio.h>
int main()
{
    int a, b;
    a=10;
    b=12;
    printf(“%d + %d = %d”, a, b, a+b);
```

```
    system(“pause”);
}
```

Оваа програма на екран ќе испечати:

10 + 12 = 22

Вежба-2

Програмски јазик C –Вежба 2

Наредби за внес и печатење на податоци

1. Внес од тастатура со користење на функцијата scanf()

Example:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
     int a;
     scanf(“%d”, &a);
     printf(“Vrednosta na promenlivata a e %d\n”,
a);
     system(“pause”);
}
```

Доколку се изврши горенаведената програма по внесување на вредноста 10 на екран се печати:

Vrednosta na promenlivata a e 10

2. Збир на два цели броја

Example:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int a, b, zbir;
    printf(“Programa za zbir na dva broja\n”);
    printf(“Vesete go prviot sobirok\n”);
    scanf(“%d”, &a);
    printf(“Vesete go vtoriot sobirok\n”);
    scanf(“%d”, &b);
    zbir = a + b;
    printf(“Zbirot na broevite %d i %d e %d\n”, a,
b, zbir);
    system(“pause”);
}
```

Доколку се изврши горенаведената програма по внесување на вредностите 10 и 5 на екран се печати:

Zbirot na broevite 10 и 5 е 15

3. Операции на збир, разлика и множење на два цели броја

Example:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int a, b;
```

```
    printf(“Vesete go prviot broj\n”);
    scanf(“%d”, &a);
    printf(“Vesete go vtoriot broj\n”);
    scanf(“%d”, &b);
    printf(“Zbirot na broevite %d i %d e %d\n”, a,
b, a+b);
    printf(“Razlikata na broevite %d i %d e %d\n”,
a, b, a-b);
    printf(“Proizvodot na broevite %d i %d e
%d\n”, a, b, a*b);
    system(“pause”);
}
```

Доколку се изврши горенаведената програма по внесување на вредностите 10 и 5 на екран се печати:

Zbirot na broevite 10 и 5 e 15

Razlikata na broevite 10 и 5 e 5

Proizvodot na broevite 10 и 5 e 50

4. Тип на променливи float

Example:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    float a, b;
    printf(“Vesete go prviot broj\n”);
```

```
    scanf(“%f”, &a);
    printf(“Vesete go vtoriot broj. Toj treba da
bide razlicen od 0\n”);
    scanf(“%f”, &b);
    printf(“Kolicnikot na broevite %f i %f e
%f\n”, a, b, a/b);
    system(“pause”);
}
```

Доколку се изврши горенаведената програма по внесување на вредностите 10 и 8 на екран се печати:

Kolicnikot na broevite 10.0 и 8.0 e 1.25

Note:Заблешка: Внимавајте на типот на променливите и специјалните знаци %f.

Наредби за разгранување
За претставување на алгоритми со разгранета структура, во програмскиот јазик С се користи наредбата if или нејзиниот облик if...else.

Наредби за разгранување

Структура за разгранување (if … else)

If структурата овозможува разгранување на текот на програмата преку избор од една или од две можности.

Ако if стуктурата се користи за избор од една можност, тогаш доколку е исполнет зададениот услов (1), се извршува наредбата напишана по if делот, во спротивно, доколку не е исполнет условот (0) се извршува следната наредба во програмата (што не е дел од if структурата).

Условот во if стуктурата **задолжително** се пишува во мали загради. За разлика од некои други програмски јазици, тука не постои клучниот збор **then**.

Ако треба да се изврши само една наредба, таа едноставно се пишува по if делот. Обликот на оваа наредба е:

```
if (логички услов)наредба;
```

Доколку пак треба да се извршат повеќе наредби (блок наредби) по if делот, тие се ставаат во блок ограничен со големи загради **{ }** и наредбата има облик:

```
if (логички услов)
  {
    наредби;
  }
```

Example:
Пример 1.

```
scanf (“%i”, &i);

if (i>0)
     printf (“Vnesen e pozitiven broj\n”);
if (i<0)
{
     printf (“Vnesen e negativen broj\n”);
     i = -1;
}
```

ВАЖНО!!!

Доколку по грешка се напише знакот **;** по if делот, тоа значи **“празна наредба”** (наредба што не врши ништо) и if структурата тука завршува, односно **не го врши разгранувањето**. Наредбата што треба да биде дел од if стуктурата, сега е надвор од неа и ќе се изврши во секој случај, а не само кога условот е точен. Тоа го илустрира следниов пример :

Example:
Пример 2.

```
printf (“Vnesete cel broj\n”);
scanf (“%i”, &i);
if (i>0);
     printf (“Vnesen e pozitiven broj\n”);
```

Оваа програма на екранот ќе печати:
Vnesete cel broj
-8

Vnesen e pozitiven broj

Ако if стуктурата се користи за избор од две можности, тогаш се користи и делот else од стуктурата if … else. Ако е исполнет зададениот услов (1), се извршува наредбата напишана по if делот, во спротивно, доколку не е исполнет условот (0) се извршува наредбата од делот else. И во овој случај доколку има по една наредба по if и else деловите таа не мора да се стави во големи загради **{ }**. Ако треба да се напишат повеќе наредби, тие се ставаат во блок ограничен со големи загради **{ }**.

Example:
Пример 3.

```
scanf (“%i”, &i);
if (i>0)
    printf (“Vnesen e pozitiven broj\n”);
else
    printf (“Vnesen e negativen broj\n”);
if (i<0)
{
    printf (“Vnesen e negativen broj\n”);
    i = -1;
}
else
{
    printf (“Vnesen e pozitiven broj\n”);
    i = +1;
}
```

Во програмскиот јазик C може да се вгнездуваат if стуктури. При вгнездувањето на if структурите, треба да се внимава за која if наредба и одговара соодветното else. По дефиниција, else се „врзува“ за најблиското if „над“ него.

Example:
Пример 4.

```
if  ( i > 0 )
   printf (“Vnesen e pozitiven broj\n”);
else
   if (i > -100)
         printf (“Vnesen e golem negativen
broj\n”);
   else
         printf (“Vnesen e mal negativen broj\n”);
```

ВАЖНО!!!

За да се провери дали една променлива припаѓа во даден интервал, задолжително во условот во if структурата треба да се користи логички оператор.

Така на пример, за да се провери дали оценката на еден студент е помала или еднаква на 10, а поголема или еднаква на 6 се користи if структурата која го користи логичкиот оператор && (ова е логичкото И) :

```
if(10 >= ocena && ocena >= 6)
```

Но **НИКАКО** не треба да се користи следната if структура:

```
if(10 >= ocena >= 6)// GRESHKA
```

Изразот 10 >= ocena >= 6 се пресметува на следниот начин: (10 >= ocena) >= 6 и значи: „Провери дали ocena е помало или еднакво на 10, а потоа провери дали добиениот резултат е поголем или еднаков на 6“.

Example:
Пример 5.
Што значи следната наредба, доколку n е кој било број :

```
if(n)
    prosek = suma / n;
```

Со оваа наредба е потребно да се пресмета просекот, само доколку n е број различен од нула (делење со 0 не смее да се дозволи).
Тривијалниот услов (израз): n секогаш се интерпретира како “точно” колку вредноста на n е „ненулта“, односно како „неточно“ доколку n е нула.
Поради истата причина се пишуваат и следните услови во if структурите:
if(izraz) наместо if(izraz != 0)
и
if(!izraz) наместо if(izraz == 0).

Задачи со разгранување
Решени задачи со раздранета структура за чија реализација се користат структурите if и if...else.

Задачи со разгранување

Example:
Пример 1.
Што ќе отпечати следниот програмски сегмент:

```
int x = 3;
if (x)
    printf ("DA\n");
else printf ("NE\n");
```

Одговор:
Ќе отпечати DA поради тоа што x има „ненулта" вредност.

Example:
Пример 2.
Под кои услови ќе се отпечати зборот Voda во следниот програмски сегмент?

```
if(T < 0)
   printf("Mraz\n");
else
   if(T < 100)
        printf("Voda\n");
   else
        printf("Parea\n");
```

Одговор:

Доколку Т е **поголемо или еднакво** на 0 **и помало** од 100. Да се внимава на ова **поголемо или еднакво**.

Example:
Пример 3.
Да се напише програма што врши квантификација на внесениот цел број: Се внесува цел број (x), тој се проверува и се печати соодветниот текст што го опишува бројот, според следната табела:
ако x е поголем или еднаков на 1000 се печати “претерано позитивен”
ако x е помеѓу 999 и 100 (вклучувајќи 100) се печати “многу позитивен”
ако x е помеѓу 100 и 0 (без 0) се печати “позитивен”
ако x е еднаков на 0 се печати “нула”
ако x е помеѓу 0 и -100 се печати “негативен”
ако x е помеѓу -100 и -999 (вклучувајќи -100) се печати “многу негативен”
ако x е помал или еднаков на -1000 се печати “претерано негативен”
На пример, за -10 ќе печати “негативен”, за -100 ќе печати “ многу негативен”, а за 458 печати “многу позитивен”
Програма:

```
#include <stdio.h>

int main ()
{
    int i;
    printf (“Vnesete cel broj: \n”);
    scanf (“%i”, &i);

    if (i>=1000 || i<= -1000 )
              printf (“preterano ”);
    else
         if (i>=100 || i<= -100 )
              printf (“mnogu ”);
    if (i>0)
```

```
              printf (“pozitiven\n”);
    else
         if (i==0)
              printf (“nula\n”);
    else  if (i<0)
              printf (“negativen\n”);

  return 0;
}
```

Example:
Пример 4.
Да се напише програма што ќе ги генерира оценките врз основа на освоените поени од испитот, според следната табела:
поени: 0-50 оцена: 5
поени: 51-60 оцена: 6
поени: 61-70 оцена: 7
поени: 71-80 оцена: 8
поени: 81-90 оцена: 9
поени: 91-100 оцена: 10
Програма:

```
#include <stdio.h>

void main ()

{
    int  i, ocena;
    printf (“Vnesete poeni: \n”);
    scanf (“%d”, &i);

    ocena = 0;
    if (i>=0 && i<=50 )
           ocena = 5;
```

```
    else
         if (i>50 && i<=60 )
            ocena = 6;
         else
              if (i>60 && i<=70 )
                   ocena = 7;
              else
                  if (i>70 && i<=80 )
                       ocena = 8;
                  else
                       if (i>80 && i<=90 )
                           ocena = 9;
                       else
                            if (i>90 && i<=100 )
                                  ocena = 10;
                            else
                              printf (“Vnesen e 
pogreshen broj za poenite!!\n”);
    if (ocena)
         printf (“Studentot dobil ocena %d.\n”, 
ocena);
}
```

Example:
Пример 5.
Престапна година е секоја година што е деллива со 4, освен доколку годината е деллива со 100 но не со 400. Да се напише програма што за година внесена од тастатура, ќе отпечати информација дали годината е престапна.
Програма:

```
#include <stdio.h>
int main ()
{
```

```
    int  godina;
    printf ("Vnesete godina: \n");
    scanf ("%d", &godina);

    if ((godina%400==0)||((godina%4==0)&&
(godina%100!= 0)))
          printf ("Godinata E prestapna: \n");
    else
          printf ("Godinata NE prestapna: \n");
    return 0;
}
```

Example:
Задача 6.
Од тастатура се внесуваат координати на една точка во рамнина. Да се напише програма што ќе одреди од кој квадрант е внесената точка или ќе даде информација доколку се работи за точка од оските.
Програма:

```
#include <stdio.h>
int main ()
{
     float  x,y;
     printf ("Vnesete kootdinati x i y: \n");
     scanf ("%f %f", &x, &y);

     if(x > 0)
     {
        if(y > 0)
               printf("Tockata e od prv
kvadrant.\n");
        else if(y < 0)
                    printf("Tockata e od cetvrti
kvadrant.\n");
```

```
            else printf("Tockata e na pozitivniot
del od x-oska.\n");
     }
     else if(x < 0)
           {
             if(y > 0)
                      printf("Tockata e od vtor
kvadrant.\n");
             else if(y < 0)
                      printf("Tockata e od tret
kvadrant.\n");
                  else printf("Tockata e na
negativniot del od x-oska.\n");
           }
           else
           {
                 if(y > 0)
                         printf("Tockata e na
pozitivniot del od y-oska.\n");
                 else if(y < 0)
                         printf("Tockata e na
negativniot del od y-oska.\n");
                      else printf("Tockata e
koordinaten pochetok\n");
           }
     return 0;
}
```

Наредби за повторување-циклуси
За контрола на тек на програма, за повроруваање на една или повеќе наредби се додека е исполнет зададен услов, се користат три вида на наредби. Тоа се наредбите while, do...while и for наредбата. Секоја од наведените наредби има своја структура и синтакса и преку примери ќе се излустрираат овие наредби.

Наредби за повторување-циклуси

Структура на наредбите за повторување, циклуси (while и do. . . while)

While структурата за повторување е една од трите структури за повторување во програмскиот јазик C и воедно таа е наједноставна. Со оваа наредба се овозможува повторување на една или повеќе наредби.

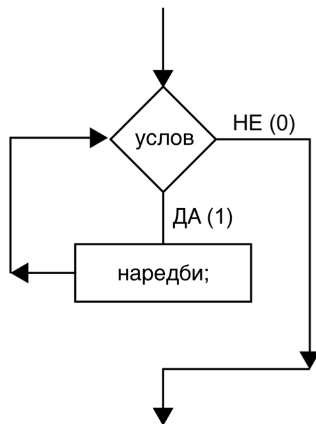
Слично како и кај if стуктурата, така и во while структурата се задава услов (**задолжително во мали загради ( )**) кој што се евалуира, односно проверува. Наредбата (наредбите) од циклусот се извршуваат се додека условот од while структурата е точен (враќа ненулта вредност). Повторувањето на наредбите од циклусот завршува кога условот ќе врати неточна вредност (нула 0) и програмата продолжува со извршување на следната наредба од телото на програмата (што не е дел од while структурата).

Ако треба да се изврши само една наредба, таа едноставно се пишува по while делот:

```
while (услов)
            наредба од циклусот;
```

а ако пак треба да се извршат повеќе наредби (блок наредби) по while делот, тие се ставаат во блок ограничен со големи загради **{ }**, односно :

```
while (услов)
{
        една или повеќе наредби;
}
```



Слика 1. Блок дијаграм на while структура

Example:

Пример 1.

Во овој пример се користи само една наредба во циклусот:

```
int j = 5;

//primer so samo edna naredba vo ciklus

while (j>0)
```

```
    printf(“j = %d\n”,j--);
```

Оваа програма ќе печати броеви, почнувајки од 5, при што во секој циклус тие се намалуваат за 1 и на екранот ќе се отпечати:
j = 5
j = 4
j = 3
j = 2
j = 1
Истиот пример може да се запише и со две наредби во циклусот, при што ќе се добие идентичен резултат:

```
int j=5;

//primer so poveke naredbi vo ciklus

while(j > 0)
{
    printf(“j = %d\n”,j);
    j--;
}
```

Доколку по грешка се напише знакот **;** по while делот со условот, наредбата/наредбите од while структурата стануваат дел “**неправи ништо**”. Бидејќи условот останува непроменет програмата влегува во бесконечна јамка (**loop**). Во ваков случај програмата нема да врати резултат, бидејќи е зафатена со “неправењето ништо”, поради што следните наредби по while структурата никогаш не се извршуваат.

Example:
Пример 2.

Пример кога while структурата неправи ништо

```
int j = 5;
while (j>0);
printf("j = %d\n",j--);
```

Во овој пример заради знакот ; во наредбата

```
while (j>0);
```

таа ништо не извршува и на екранот ќе нема никаков одговор.

Многу е важно да се нагласи дека сите услови се while услови, односно "**се додека е исполнет условот се извршуваат наредбите**". Наредбите од циклусот while се извршуваат се додека е точен условот (има вредност различна од 0). Затоа, за да не се влезе во бесконечна јамка, во наредбите што седуваат после while наредбата мора да постои наредба која ќе влијае на вредноста на условот во наредбата while.

Example:
Пример 3.
Во овој пример, програмерот најверојатно имал намера наредбата од while структурата да се извршува се додека j не стане еднакво на 0. Но направена е **грешка** во поставување на условот, точен услов би бил се додека j е различно од 0 (j != 0).
Затоа програмата:

```
int j = 5;
printf("start\n");
while (j == 0)
     printf("j = %d\n",j--);
printf("end\n");
```

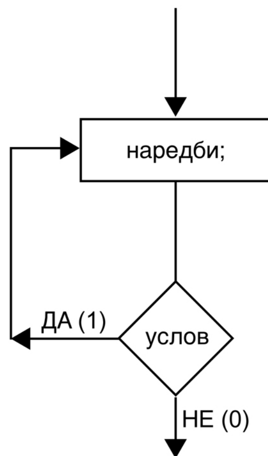
нема да ја извршува наредбата од while циклусот, туку само ќе отпечати:
start
end

Друг облик на структура за повторување е **do. . . while** структурата која е “превртена” верзија на структурата за повторување while. Ако во while циклусот по условот следува телото со наредби што се повторуваат, кај do. . . while циклусот прво се пишува телото со наредби за повторување, па следува условот. Од ова произлегува дека наредбите од циклусот ќе бидат извршени сигурно еднаш, пред да се провери условот. Ако условот е неточен, наредбите од циклусот do. . . while нема да бидат повторно извршени. Обликот на оваа структура е

```
do
   тело (наредба за повторување);
while (услов);
```

или пак

```
do
{
   блок наредби;
}
while (услов);
```



Слика 2. Блок дијаграм на структурата do … while

Example:

Пример 4.

Користејќи ја do…while структурата, програмата од Примерот 1 сега може да се запише како:

```
int j = 5;
printf("start\n");
do
    printf("j = %d\n",j--);
while (j > 0);
printf("stop\n");
```

и на екранот ќе се отпечати:

start
j = 5
j = 4

j = 3
j = 2
j = 1
stop

Example:
Пример 5.
Програмата

```
int j = -10;
printf("start\n");
do
{
    printf("j = %d\n",);
    j--;
}
while (j > 0);
printf("stop\n");
```

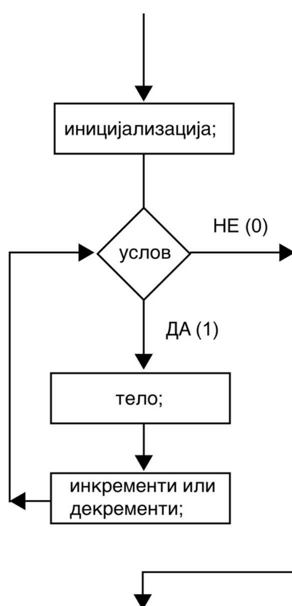
на екранот ќе отпечати само
start
j = -10
stop

Структура за повторување (for)

Од аспект на синтаксата **for** циклусот за повторување е најсложен. Сепак, тој е сличен на **while** циклусот, бидејќи и самиот содржи while тип на услов (се додека е исполнет, односно точен условот се извршуваат наредбите од телото). Синтаксата на оваа структура за повторување е од облик:

```
for(иницијализација; услови; инкременти или
декременти)
       тело (наредба/наредби за повторување);
```

[missing_resource: draw_odg0.png]



Слика 3. Блок дијаграм на структурата for

Наредбите од делот **иницијализација** се извршуваат точно еднаш, на почетокот - пред влезот во циклусот. Наредбите од делот **услови** се извршуваат пред почетокот на секој нов циклус и доколку тие имаат вредност која се интерпретира како логичка вистина се влегува во повторување на наредбите од циклусот, во спротивно се завршува повторувањето на циклусот и се продолжува со наредбите што следуваат по циклусот во програмата. Наредбите од делот **инкременти или декременти** се извршуваат на крајот на секој циклус (по

извршувањето на сите наредби од **телото** на циклусот) по што се извршуваат наредбите од делот **услови** и доколку се задоволени, циклусот се повторува.

Исто како и кај if и while, доколку треба да се изврши само една наредба, таа едноставно се пишува по for делот, доколку, пак, треба да се извршат повеќе наредби (блок наредби) по for делот, тие се ставаат во блок ограничен со големи загради **{}**.

ВАЖНО!!!

Да се внимава дека знакот **;** мора да ги одделува трите дела на **for** наредбата. Исто така условот во for наредбата мора да биде од типот "**се додека е исполнет условот**".

Example:
Пример 6.
Истиот пример, наведен претходно за печатење на почетниот број 5 и потоа во секој циклус тој се намалува за 1 и се печати бројот се додека се добива позитивен број, е прикажан и на трет начин со структурата for:

```
int j;

for (j = 5; j > 0; j--)
      printf("j = %d\n",j--);
```

и резултатот што ќе се отпечати на екранот е:
j = 5
j = 4
j = 3
j = 2
j = 1

Треба да се забележи дека за разлика од некои програмски јазици, каде чекорот во for циклусот можеше да биде само 1, во C не постои ограничување за чекорот.

Example:
Пример 7.
Програма за табелирање на функцијата sin$x$,$0 \leq x \leq \pi$ со чекор 0.2.

```
#include <stdio.h>
#include <math.h>
int main()
{
     double agol;
     for(agol = 0.0; agol < 3.14159; agol += 0.2)
          printf("sinus od %.1lf e %.2lf\n", agol,
sin(agol));
     return 0;
}
```

Во овој пример, променливата agol се зголемува за 0.2 радијани при секое извршување на наредбата for, се додека е исполнет условот agol < 3.14159. Како што е наведено, вредностите на аглите задолжително се задаваат со радијански мерки.

Example:
Пример 8.
Во деловите за иницијализација и менување на вредности од for наредбата, по потреба може да се наведат повеќе изрази кои се одвојуваат со знакот запирка(**,**). Запирката гарантира последователно извршување на наведените изрази.

```
int i, j, k;
    for (i = 0, j = 5, k = -1; i < 10; i++, j++,
```

```
k--)
```

Ако нема потреба од наведување на изрази во овие два дела, тие можат да бидат изоставени, но **НЕ СМЕАТ** да се заборават знаците **;** . Коректно напишана е следната наредба

```
for(; i < 10; i++, j++, k--)
```

Со наредбата

```
for( ; ; )
```

се креира бесконечна јамка и се чита “засекогаш”(for ever).

Клучни зборови break и continue

Од претходните примери за наредбите while и for се гледа можноста за креирање на бесконечен циклус. Знаејќи го ефектот од користењето на бесконечниот циклус произлегува дека треба да се одбегнува по секоја цена. Сепак, во програмскиот јазик C може да се користат бесконечни циклуси од кои ќе се излегува, односно скокнува надвор од нив. Користењето на клучниот збор **break** овозможува излегување од било кој циклус, независно од условот и продолжување со првата следна наредба по циклусот.

Example:
Пример 9.

```
for(;;)
{
   printf(“vnesi int vrednost:”);
   if(scanf(“%d”, &j) == 1)
        break;
```

```
   while((c = getchar()) != ‘\n’);
}
printf(“j = %d\n”, j);
```

Оваа програма на екранот ќе испечати:
vnesi int vrednost: an int
vnesi int vrednost: no
vnesi int vrednost: 16
j = 16
Во случај ако scanf врати 1 (односно ако се вчита преку тастатура цел број), се излегува од бесконечната јамка.

Пример 10.

Example:
Пример 10.

```
for(j = 1; j <= 10; j++)
{
   if(j % 3 == 0)
       continue;
   printf(“j = %d\n”, j);
}
```

На екранот оваа програма ќе печати:
j = 1
j = 2
j = 4
j = 5
j = 7
j = 8

```
j = 10
```

Задачи со циклуси
Се прикажуваат повеќе задачи за чија програмска реализација се користат структури за повторување-циклуси.

Example:
Пример 1.
Да се состави програма која ќе ги испечати првите 10 природни броја.
Програма:

```
#include <stdio.h>

main()
{
     int i=1;

     while (i <= 10)
     {
          printf ("i e %i\n", i);
          i++
     }
}
```

Example:
Пример 2.
Да се испечатат сите едноцифрени парни броеви.
Програма:

```
#include <stdio.h>

main()
{
     int i=1;
```

```
     while (i <= 10)
     {
         if (i%2==0) printf ("i e %i\n", i);
         i++
     }
}
```

Example:
Пример 3.
Да се состави програма која ќе ја пресмета средната вредност на пет броеви кои се внесуваат преку тастатура.
Програма:

```
#include <stdio.h>

int main()
{
     int n = 1;
     int broj, suma= 0;
     while( n <= 5 )
     {
          printf( "Vnesi broj: ");
          scanf("%d", &broj);
          suma+= broj;
          n++;
     }
     printf("\nSredna vrednost na vnesenite broevi
"
     “e %f\n", (float)suma/(n-1));
     return(0);
}
```

Example:
Пример 4.
Да се испечатат сите броеви деливи со 3 и помали од 1000. Да се преброи колку такви броеви има:
Програма:

```
#include<stdio.h>

int main()
{
     int n = 1;
     int vkupno = 0;
     while( n <= 1000 )
     {
          if (n%3==0)
                {
                     printf("%d\n",n);
                     vkupno++;
                }
     }
     printf("Vlupno takvi broevi ima
%d\n",vkupno);
     return(0);
}
```

Example:
Пример 5.
Да се напише програма за пресметување на $y = x^n$ за даден природен број n и релаен број x.
Програма:

```
#include<stdio.h>
```

```
int main()
{
     int brojac, n;
     float x, y;
     printf(“vnesi vrednost za osnovata:
“);scanf(“%f”, &x);
     printf(“vnesi vrednost za eksponentot:
“);scanf(“%d”, &n);
     y = x;
     brojac = 1;
     while (brojac < n)
     {
         y *= x;
         brojac++;
     }
     printf(“%f^%d = %f\n”, x, n, y);
return 0;
}
```

Example:
Пример 6.
Да се напише програма која од n броеви (внесени од тастатура) ќе го определи бројот на броеви што се деливи со 3, како и бројот на броеви за кои при делењето со 3 имаат остаток 1, односно 2.
Програма:

```
#include<stdio.h>

int main()
{
     int  n, i, broj, del, os1, os2;
     del = os1 = os2 = 0;
     printf(“Kolku broevi treba da se proveruvaat
```

```
za delivost so 3?\n”);
     scanf(“%d”, &n);
     i = 1;
     do
     {
          printf(“Vnesete broj za proverka: “);
          scanf(“%d”, &broj);
          if ( broj % 3 == 0) del++;
          else if ( broj % 3 == 1) os1++;
               else os2++;
          i++;
     }
     while (i <= n);
     printf(“%d broevi se delivi so 3.\n“, del);
     printf(“%d broevi imaat ostatok 1, pri 
delenje so 3.\n“, os1);
     printf(“%d broevi imaat ostatok 2, pri 
delenje so 3.\n“, os2);
return 0;
}
```

Example:
Пример 7.
Да се напише програма која за даден природен број m ќе ги испише сите броеви помеѓу 1 и 1000 за кои сумата на цифрите изнесува m.
Програма:

```
#include <stdio.h>

int main ()
{
      int  m, i, j, pom, suma, cifra;
```

```
      printf(“Vnesete go brojot m za sporedba: “);
      scanf(“%d”, &m);
      if (m > 27) printf(“Ne postoi broj od 1 do
1000 so suma %d.\n”, m);
      else
      {
           printf(“Broevi cij zbir na cifri e %d
se: \n”, m);
           i = 1;
           while (i<=1000)
           {
                pom = i; suma = 0;
                while (n > 0)
                {
                     cifra = pom % 10;
                     suma += cifra;
                     pom /= 10;
                }
                if (suma == m) printf(“%d\t”, i);
                i++;
           }
      }
      return 0;
}
```

Note:
ВАЖНА ЗАБЕЛЕШКА!!!
Да се нагласи зошто е потребно користењето на променливата pom.

Example:
Пример 8.

Да се испечатат сите броеви од 1000 до 100000 кои се деливи со 9.
Програма:

```
#include <stdio.h>

int main ()
{
      int  i;
      for(i=1000; i < 10000; i++)
           if (i%9==0)
                 printf("%d\n", i);
      return 0;
}
```

Example:
Пример 9.
Да се испечатат и избројат сите природни броеви деливи со 3 чиј квадрат е помал од 1000.
Програма:

```
#include <stdio.h>

int main ()
{
     int i;
     int vkupno = 0;
     for(i=1; i*i < 1000; i++)
               if (i%3==0)
               {
                    printf("%d\n", i);
                    vkupno++;
               }
           printf("Vkupno broevi ima %d\n",
vkupno);
```

```
     return 0;
}
```

Example:
Пример 10.
Сите целобројни вредности за Целзиусови степени од 0 до 100 да се претстават во Келвини.
Програма:

```
#include <stdio.h>

int main ()
{
     int i;
     for(i=1; i <= 100; i++)
            printf("%d\n", i+273);
     return 0;
}
```

Example:
Пример 11.
Да се напише програма која ќе ги испечати сите броеви од зададен опсег кои исто се читаат и од лево на десно и од десно на лево.
Програма:

```
#include <stdio.h>

int main ()
{
      int  i, odb, dob, pom, prev, cifra;
      printf("Vnesete vrednost za opsegot.\n");
      printf("Od koj broj?\n"); scanf("%d", &odb);
```

```
      printf("Do koj broj?\n"); scanf("%d", &dob);
      for (i = odb; i <= dob; i++)
      {
             pom = i;
             prev = 0;
                  while (pom > 0)
                  {
                       cifra = pom % 10;
                       prev=prev*10 + cifra;
                       pom /= 10;
                  }
                  if (prev == i) printf("%d\t",
i);

      }
      return 0;
}
```

Example:
Пример 12.
Да се напише програма која ќе ги испечати сите броеви помали од број N составени само од парни цифри и ќе врати колку такви броја има.
Програма:

```
#include <stdio.h>

int main ()
{
      int  i, n, pom, prov, cifra, brojac;
      printf("Do koj broj treba da se proveruva?
\n");
      scanf("%d", &n);
      brojac = 0;
```

```
	for (i = 1; i <= n; i++)
	{
		prov = 1;
		pom = i;
		while ((pom > 0) && prov)
		{
			cifra = pom % 10;
			if (cifra % 2) prov = 0;
			pom /= 10;
		}
		if (prov)
		{
			printf("%d\t", i);
			brojac++;
		}
	}
	printf("\nIma vkupno %d broevi so samo parni cifri\n", brojac);
	return 0;
}
```

Низи-едноиндексни полиња
Низата како едноиндексно поле се користи и е погодна за пресметки со прообемни податоци бидејќи податоците во низа имаат исто име, а се разликуваат само по својата позиција-индекс кој е природен или цел број. Тоа овозможува лесно оперирање со податоците од низите бидејќи во пресметките се оперира само со нивните индекси.

Низи-едноиндексни полиња

Што е низа во програмските јазици?

- **Низите се структурирани податочни типови**
- **колекција од повеќе индивидуални елементи со заедничко име**
- **овозможуваат пристап до индивидуалните елементи во низата**

За да се организираат повеќе податоци од ист тип, може да се креираат повеќе променливи и на секој податок му се доделува промелива. Но тоа е фукционално само ако се оперира со ограничен број променливи.

На пример, со програмскиот сегмент се собираат вредностите на три податока

```
int  prom1,  prom2,  prom3;
int  suma;
scanf(“%d%d%d”,&prom1,&prom2,&prom3);
suma =  prom1 + prom2 + prom3;
```

Но ако е потребно да оперираме со многу повеќе променливи од исти тип (на пр. 100, 1000, ...), потребен е поинаков приод. За таа цел се формира низа од повеќе променливи од ист тип.

Низа е структура со релациски поврзани податоци, при што

- бројот на елементи е однапред познат
- сите елементи се променливи од ист тип

Дефиниција:

Низа е колекција од променливи од ист тип сместени во низа последователни мемориски локации, на кои им е доделено единствено име.

Низите уште се нарекуваат едноиндексни или еднодимензионални полиња.

За пристап до кој и да е елемент од полето се користи името и позицијата на елементот во полето, а позицијата се одредува со индекс.

Според бројот на индекси полињата се делат на

- едноиндексни (полиња),
- двоиндексни (матрици), итн..

Име на поле
сите елементи имаат
заедничко име c

c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1
c[10]	6453
c[11]	78

Број (индекс) - ја одредува
позицијата на елементот во
полето c

Декларирање на едноиндексно поле

Секогаш кога во програма се користи поле, на почетокот од програмата тоа се декларира. За негово декларирање потребно е неговото име, типот на елементите (променливите) и бројот на елементи во полето.

Формат на наредбата за декларирање е

```
tip ImePole[BrojNaElementi];
```

Примери за декларирање на поле:

```
int c[10];  float moePole[3284]; double Tez[100];
int b[100], x[27];
```

Example:
Пример 1.
Да се дефинира едноиндексно поле temps што содржи 5 реални вредности.
Со наредбата

```
float  temps[5];
```

се декларира поле temps со пет реални вредности, а тоа значи резервирање мемориски простор



Иницијализација на едноиндексно поле

Во наребата за декларација може да се иницијализираат и елементите на полето како што е наведено со следните две наредби:

```
int p[10] ={1, 2, 3, 4, 5, 6, 7, 8, 9, 10}, n[5] =
{1, 2, 3, 4, 5 };

float broevi[4] = {1.0, 0.3, 2.25, 4.5};
```

Ако не се внесат доволно вредности за сите елементи од полето, најдесните елементи се иницијализираат на 0 како на пример:

```
int a[5] = {0,1,3};   /* preostanatite elementi se
0 */

int b[10]={0};  /*najednostaven nacin za
inicijalizacija na site vrednosti na 0 */

int c[80] = {95};  /* prviot element ima vrednost
95, a site ostanati 0 */
```

Ако не е определена големината на полето, тогаш истата ја определува компјутерот

int n[] = { 1, 2, 3, 4, 5 }; - 5 почетни вредности, согласно полето ќе биде декларирано како поле со 5 елемента

int pogodi[] = {1,3,5,7,11,13}; - поле со 6 елементи

int golemina[] = {3, 1, 5, 99, 18, -1}; - поле со 6 елементи

Пристап до елемент на едноиндексно поле

За да се добие или промени елемент од полето, потребно е да се определи име на полето и позицијата на елементот во полето. Форматот на елемент од полето е

```
imePole[IndeksPozicija]
```

Поле со n елементи и име alfa е:

alfa[0], alfa[1]...alfa[n-1]

- првиот елемент се наоѓа на позиција (има индекс) 0, вториот на позиција (има индекс) 1, итн.
- индекс на поле може да биде само целобројна вредност
- одговорност на програмерот е да обезбеди индексот да биде во интервалот [0,n-1]

Поле може да се иницијализира и со следната низа наредби:

```
int i, n[10];

for (i=0; i<10; i++) n[i] = i;
```

Example:
Пример 2.
Програмски сегмент за собирање на вредностите на елементите на две полиња и нивно сместување во трето поле:

```
for(i=0; i<n; i++)
c[i]=a[i]+b[i];
```

Example:
Пример 3.
Операторот што го одредува индексот има најголем приоритет, поради што во следниот израз

```
a[2]++
```

компјутерот ќе ја зголеми вредноста на променливата што се наоѓа на третата позиција во полето за 1.

Example:
Пример 4.
Да се најде минимален елемент во низа.
Програма:

```
#include <stdio.h>

int main()
{
     int min, i, pole[20];
     for(i = 0; i < 20; i++)
     {
           printf(“Vnesete element %d:i”, i);
           scanf(“%d”, &pole[i]);
     }
     min = pole[0];
     for(i = 0; i < 20; i++)
     {
           if (pole[i]<min) min = pole[i];
     }

     printf(“Minimalen e elementot %d ”, min);
     return 0;
}
```

Example:
Пример 5.

Да се изброи колку елементи од една низа припаѓаат на даден интервал.

Програма:

```
#include <stdio.h>

int main()
{
      int pocetok, kraj, i, pole[50], brojac = 0;
      printf (“Vnesi vrednoti za pocetok i kraj na
intervalot”);
      scanf (“%d %d”, &pocetok, &kraj);

      for(i = 0; i < 50; i++)
      {
              printf(“Vnesete element %d:i”, i);
              scanf(“%d”, &pole[i]);
      }
      for(i = 0; i < 50; i++)
      {
              if (pole[i]>pocetok && pole[i]<kraj)
brojac++;
      }

      printf(“Vkupno elementi vo intervalot ima %d
”, brojac);
      return 0;
}
```

Програми со низи
Се прикажуваат повеке задачи со низи и нивни решенија во вид на програма во програмскиот јазик "C".

Example:
Пример 1.
Од дадена низа да се формира подниза од елементи чии вредности се од даден интервал кој ќе се внесе преку тастатура. Да се отпечати новата низа.
Програма:

```
#include <stdio.h>
int main()
{
      int pocetok, kraj, vlez[50], izlez[50];
      int brojac = 0, i;
      printf (“Vnesi pocetok i kraj”);
      scanf (“%d %d”, &pocetok, &kraj);

      for(i = 0; i < 50; i++)
      {
            printf(“Vnesete element %d:i”, i);
            scanf(“%d”, &vlez[i]);
      }

      for(i = 0; i < 50; i++)
      {
            if (vlez[i]>pocetok && vlez[i]<kraj)
            {
                  izlez[i]=vlez[i];
                  brojac ++;
            }
      }
      printf(“Izleznata niza:”);
      for(i = 0; i < brojac; i++)
```

```
	{
		printf(“izlez[%d]:%d\n”, i, izlez[i]);
	}
	return (0)
}
```

Example:
Пример 2.
Да се прикаже бројот на деновите во сите месеци во годината.
Програма:

```
#include <stdio.h>
int main()
{
     int i, meseci[] =
{31,28,31,30,31,30,31,31,30,31,30,31};

     for(i = 1; i < 13; i++)
            printf("Mesecot broj %d ima %d
denovi\n",i,meseci[i-1]);
}
```

Example:
Програмски сегмент:
Да се напише програмски сегмент што ќе овозможи проверка дали две низи a и b со должини n1 и n2 се идентични.

Note:
Забелешка

Две низи се идентични ако имаат ист број на елементи и ако елементите на двете низи што се наоѓаат на иста позиција имаат идентична вредност.

Програмски сегмент:

```
int i;
if(n1 == n2) for(i=0; a[i]==b[i] && i<n1; i++);
if(i==n1)  Исти се ...
```

Example:

Пример 4.

Да се напише програма која за низа a составена од 10 целобројни вредности, ќе ја пресмета просечната вредност на елементите од низата.

Програма:

```
#include <stdio.h>
void main()
{
     int a[10], i;
     float suma=0.0;
     printf ("Vnesi 10 elementi od nizata:\n");
     for(i=1;i<=10;i++)
     {
          scanf("%d",&a[i]);
     }
     for(i=1;i<=10;i++)
     {
          suma+=a[i];
     }
     printf("Prosecnata vrednost e:
```

```
%f.\n",suma/10.0);
}
```

Example:

Пример 5.

Да се напише програма која за дадена низа а од m (m <100) реални броеви ќе ја најде аритметичката средина на елементите. Потоа низата да се трансформира така што ќе се исфрлат елементите помали од средната вредност, и новодобиената низа да се испечати на екран.

Програма:

```
#include <stdio.h>
void main()
{
     int i, m, j=0;
     float arSred=0.0, a[100], b[100];
     printf("Kolku elementi ima nizata (m
<100):\n");
     scanf("%d", & m);
     for( i=1; i<= m; i++ )
     {
          scanf("%f",&a[i]);
     }
     for(i=1;i<= m;i++)
     {
          arSred+=a[i];
     }
     arSred/= m ;
     for(i=1;i<= m ;i++)
          if(!(a[i]<arSred))
          {
               j++;
               b[j]=a[i];
          }
```

```
      for(i=1;i<=j;i++)
           printf("b[%d] = %f\n", i,  b[i]);
}
```

Example:
Пример 6.
Непознат број студенти одговарале на по десет прашања. На секое од прашањата студентот или не одговорил или одговорил со заокружување на еден од понудените пет одговори (1, 2, 3, 4 и 5). За секој точен одговор студентот добива по два поени, а за секој погрешен му се одзема по еден поен. Да се напише програма која на почетокот од тастатура ќе ги прочита точните одговори на десетте прашања, а потоа за внесените одговори на секој студент ќе го испечати бројот на поени кои ги освоил студентот.

Note:
Забелешка:
Прашањата на кои студентот не одговорил се кодираат со било кој знак различен од 1, 2, 3, 4 и 5.

Програма:

```
#include<stdio.h>
#define BrPrasanja 10
void main()
{
      int kluc[BrPrasanja], odgovor[BrPrasanja];
      char c=’ ‘;
      int bod, i, j=1;

      printf(“Vnesi tocni odgovori:\n”);
```

```
      for(i=1;i<=BrPrasanja;i++)
           scanf("%d",&kluc[i]);
      printf("Vnesuvaj odgovori na studenti:\n");
      while(c != '.')
      {
          for(i=1;i<=BrPrasanja;i++)
               scanf("%d",&odgovor[i]);
          bod=0;
          for(i=1;i<=BrPrasanja;i++)
               if((odgovor[i]>=1)&&(odgovor[i]
<=5))
               if(odgovor[i]==kluc[i]) bod+=2;
               else bod-=1;
          printf("Student %d: %d bodovi.\n", j,
bod);
          j++;
          printf("Sleden student (. Za kraj)\n");
          c=getchar();
       }
}
```

Example:
Пример 7.
Да се напише програма која за дадена низа A од реални броеви и N, број на елементи во A, ќе ја пресмета најголемата разлика меѓу два соседни елементи од оваа низа.
Програма:

```
#include "stdio.h"
#include "math.h"
#define MAXELEM 10
void main(void)
```

```
{
     float a[MAXELEM];
     int i,n;
     float raz,maxr;

     printf(“Vnesi kolku elementi ke ima nizata
A\n”);
     scanf(“%d”,&n);
     printf(“Vnesi gi elementite na nizata A\n”);
     for (i=0; i<n; i++)
     {
          printf(“A[%d]=”,i);
          scanf(“%f”,&a[i]);
     }
     maxr=fabs(a[1]-a[0]);
     for (i=2; i<n; i++)
     {
          raz=fabs(a[i]-a[i-1]);
          if(raz>maxr)
               maxr=raz;
     }

     printf(“Najgolemata razlika medju dva sosedni
elementi e:%f\n”,maxr);
}
```

Example:
Пример 8.
Да се напише програма која во низа од N цели броеви ќе го пронајде почетокот и должината на најдолгата растечка подниза.
Програма:

```
#include "stdio.h"
#define MaxElem 10
int main()
{
	float a[MaxElem];
	int i,n,pos,len;

	printf("Dolzina na nizata: ");
scanf("%d",&n);
	for (i=0; i<n; i++)
		scanf("%f",&a[i]);

	int poc,dolz;

	poc=0; dolz=1;
	pos=0; len=1;
	i=0;
	while (i< n-1)
	{
		poc=i;
		dolz=1;
		while ((a[i] < a[i+1]) && (i < n))
		{
			i=i+1;
			dolz=dolz+1;
		}
		if (dolz>len)
		{
			len=dolz;
			pos=poc;
		}
		i=i+1;
	}

	printf("Pocetok:%d, dolzina:%d",pos,len);

	return 0;
```

```
}
```

Example:
Пример 9.
Да се напише програма во која од тастатура се чита влезната низа и потоа ќе се испечати трансформирана низа на следниот начин:
Equation:

На пример, влезната низа 1, 2, 3, 5, 7, треба да се трансформира во 8, 7, 6, 7, 8.
Програма:

```
#include “stdio.h”
#define Max 100
int main()
{
	int n,i,a[Max];

	printf("Kolku elementi ima nizata: ");
	scanf(“%d”,&n);
	printf(“Vnesi gi elementite na nizata\n”);
	for (i=0; i<n; i++)
		scanf(“%d”,&a[i]);

	int b[Max];
```

```
        for (i=0; i<n; i++)
              b[i]=a[i]+a[n-i-1];

        for (i=0; i<n; i++)
              printf(“b[%d]=%d a[%d]=%d\t”,i,*
(b+i),i,a[i]);
        printf(“\n”);

        return 0;
}
```

Example:

Пример 10.

Да се напише програма која ќе прочита од тастатурата несортирана низа А од цели броеви за потоа да ја испечати низата во ист редослед, но предходно ќе изврши бришење на броевите кои се дупликати.

На пример, за дадено поле А: 15, 31, 23, 15, 75, 23, 41, 15, 31, 85 од 10 цели броеви, излезното поле треба да биде: 15, 31, 23, 75, 41, 85 со должина 6.

Програма:

```
#include "stdio.h"
#define MAX 100

int main()
{

     int i,n,brElem, a[MAX];

     printf("Kolku elementi ima nizata: ");
     scanf("%d",&n);
```

```
     printf("Vnesi gi elementite na nizata\n");
     for (i=0; i<n; i++)
            scanf("%d",&a[i]);

     int j, brojac;

     brojac=0;
     for (i=0; i<n; i++)
     {
           j=0;
           while ((j <= brojac) && (a[i] != a[j]))
               j++;
           if (j > brojac)
           {
               brojac++;
               a[brojac]=a[i];
           }
      }
      brElem=brojac+1;

      for (i=0; i<brElem; i++)
              printf("%d",a[i]);

      return 0;
}
```

Матрици-двоиндексни полиња

Се покажува како во програмскиот јазик С се користат матриците: како се декларираат, означуваат, иницијализираат и како се пристапува до нив.

Матрици-двоиндексни полиња

Матрица или двоиндексно поле е правоаголна шема од елементи со заедничко име и два индекса кои го определуваат местото на секој елемент во матрицата преку позицијта на редот и колоната во матрицата. На пример, ознака за матрица е , каде е името на матрицата, *i* - е бројот на редици, *j* - е бројот на колони. Оваа матрица ќе има вкупно *i* x *j* елементи. Секој елемент од матрицата А во програмскиот јазик С се означува со .

	Колона 0	Колона 1	Колона 2	Колона 3
Ред 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Ред 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Ред 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Индекс на колона

Име на матрица

Индекс на ред

Матрица А (3x3)

При користење на матрица во програма се врши нејзина декларација, а во декларацијата на матрица прво се определува имато, потоа редиците, па потоа колоните во формат:

tip Ime [Redovi] [Koloni];

Иницијализација на вредностите од матрица

Матриците може да се иницијализираат со декларацијата:

```
int mat[2][2] = {{1,2},{3,4}};
```

или

```
int mat [3][4] = {
                  {100, 101, 102, 103},
                  {104, 105, 106, 107},
                  {108, 109, 110, 111}
                 };
```

Пристап до елемент од матрица

До елемент од матрица се пристапува преку неговото име и по него се задава редот, а потоа колоната во која припаѓа елементот.

Форматот е:

```
imePole[IndeksRED][IndeksKOLONA]
```

Example:
Пример 1.
Со наредбата

```
printf( "%d", b[0][1] );
```

се печати елементот b[0][1]

Example:

Пример 2.

Да се прочита и отпечати квадратна матрица (бројот на редови и колони е ист) со вредности на елементите кои ќе се внесат преку тастатура.

Програма:

```
#include <stdio.h>
#define RED 10
#define KOLONA 3

int main()
{
      int n,a[MAX][MAX], i, j;

      printf("Dimenzija: ");
      scanf("%d",&n);

      printf("Vnesi gi elementite\n");
      for (i=0; i<n; i++)
            for (j=0; j<n; j++)
                  scanf("%d",&a[i][j]);

      printf("Pecatenje\n");
      for (i=0; i<n; i++)
            {
            for (j=0; j<n; j++)
                 {
                    printf("%d\t",a[i][j]);
                 }
```

```
            printf("\n");
            }
      return 0;
}
```

Example:

Пример 3.

Да се формира матрица со елементи чии вредности се формираат преку сумата на индексите на елементот од дадената позиција.

Програма:

```
#include <stdio.h>
#define MAX 10

int main()
{

      int red, kol, pole[RED][KOLONA];

      for(red=0; red<RED; red++)
          for(kol=0; kol<KOLONA; kol++)
               pole[red][kol]=red+kol;

      for(red=0; red<RED; red++)
          for(kol=0; kol<KOLONA; kol++)
              printf(“Vektor[%d][%d] = %d”, red,
kol, vek[red][kol]);

      return 0;
}
```

Example:

Пример 4.

Да се прочита една квадратна матрица, а потоа да се помножат сите парни елементи со 2, а сите непарни со 3 и да се отпечати резултантната матрица.

Програма:

```
#include <stdio.h>
#define RED 10
#define KOLONA 3
int main()
{

      int n,a[MAX][MAX], i, j;

      printf("Dimenzija: ");
      scanf("%d",&n);

      printf("Vnesi gi elementite\n");
      for (i=0; i<n; i++)
            for (j=0; j<n; j++)
                  scanf("%d",&a[i][j]);

      for (i=0; i<n; i++)
            for (j=0; j<n; j++)
            {
                 if (a[i][j]%2==1)
                       a[i][j]*=3;
                 else
                       a[i][j]*=2;
            }


      printf("Pecatenje\n");
      for (i=0; i<n; i++)
      {
            for (j=0; j<n; j++)
```

```
                {
                     printf("%d\t",a[i][j]);
                }
                printf("\n");
          }

          return 0;
}
```

Example:
Пример 5.
Да се прочита една квадратна матрица, да се помножат сите елементи од главната дијагонала со 10, а сите елементи од споредната дијагонал да се поделат со 5 и да се отпечати резултантната матрица.
Програма:

```
#include <stdio.h>
#define RED 10
#define KOLONA 3
int main()
{

      int n, a[MAX][MAX], i, j;

      printf("Dimenzija: ");
      scanf("%d",&n);

      printf("Vnesi gi elementite\n");
      for (i=0; i<n; i++)
            for (j=0; j<n; j++)
                  scanf("%d",&a[i][j]);
```

```
	for (i=0; i<n; i++)
		for (j=0; j<n; j++)
		{
			if (i==j)
				a[i][j]*=10;
			else if (i+j==n-1)
				a[i][j]/=5;
		}

	printf("Pecatenje\n");
	for (i=0; i<n; i++)
	{
		for (j=0; j<n; j++)
		{
			printf("%d\t",a[i][j]);
		}
		 printf("\n");
	}

	return 0;
}
```

Упатство за работа за DEV CPP
Упатство за работа со компајлер во програмскиот јазик Ц.

Упатство за работа за DEV CPP

1. Се вклучува Start All Programs > Bloodshed DEV C++
2. Се оди на опцијата File > New > Project
3. Се избира Console Application, се избира C Project и се именува проектот, пример: Zadaca1 (наставка ќе си додаде и сам)
4. Потоа прашува каде да се сними, се избира фолдер по ваш избор
5. Се пишува кодот во изворниот фајл. Програмата сама генерира бланко изворна датотека од следниот облик:

```
#include <stdio.h>

#include <stdlib.h>

int main(int argc, char *argv[])

{

/* OVDE SE PISIVA VASIOT KOD */

system("PAUSE");

return 0;

}
```

1. Се компајлира и извршува со притискање на F9 или на опцијата од менито Execute > Compile & Run
2. Ако станува збор за првото извршување на новата програма, ќе ни понуди најпрво да ја сними датотеката со изворниот код, со име main.c, но треба да го преименуваме со наше ново име. Пример: zadaca1.c (наставка C ќе си додаде и сам.)